Key Concepts

Software evolution | Procedure-oriented programming | Object-oriented programming | Objects | Classes | Data abstraction | Encapsulation | Inheritance | Polymorphism | Dynamic binding | Message passing | Object-oriented languages | Object-based languages

1.1 Software Crisis

Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry. The following issues need to be addressed to face this crisis:

- How to represent real-life entities of problems in system design?
- How to design systems with open interfaces?
- How to ensure reusability and extensibility of modules?
- How to develop modules that are tolerant to any changes in future?
- How to improve software productivity and decrease software cost?
- How to improve the quality of software?
- How to manage time schedules?
- How to industrialize the software development process?

Many software products are either not finished, or not used, or else are delivered with major errors. Figure 1.1 shows the fate of the US defence software projects undertaken in the 1970s. Around 50% of the software products were never delivered, and one-third of those which were delivered were never used. It is interesting to note that only 2% were used as delivered, without being subjected to any changes. This illustrates that the software industry has a remarkably bad record in delivering products.

Changes in user requirements have always been a major problem. Another study (Fig 1.2) shows that more than 50% of the systems required modifications due to changes in user requirements and

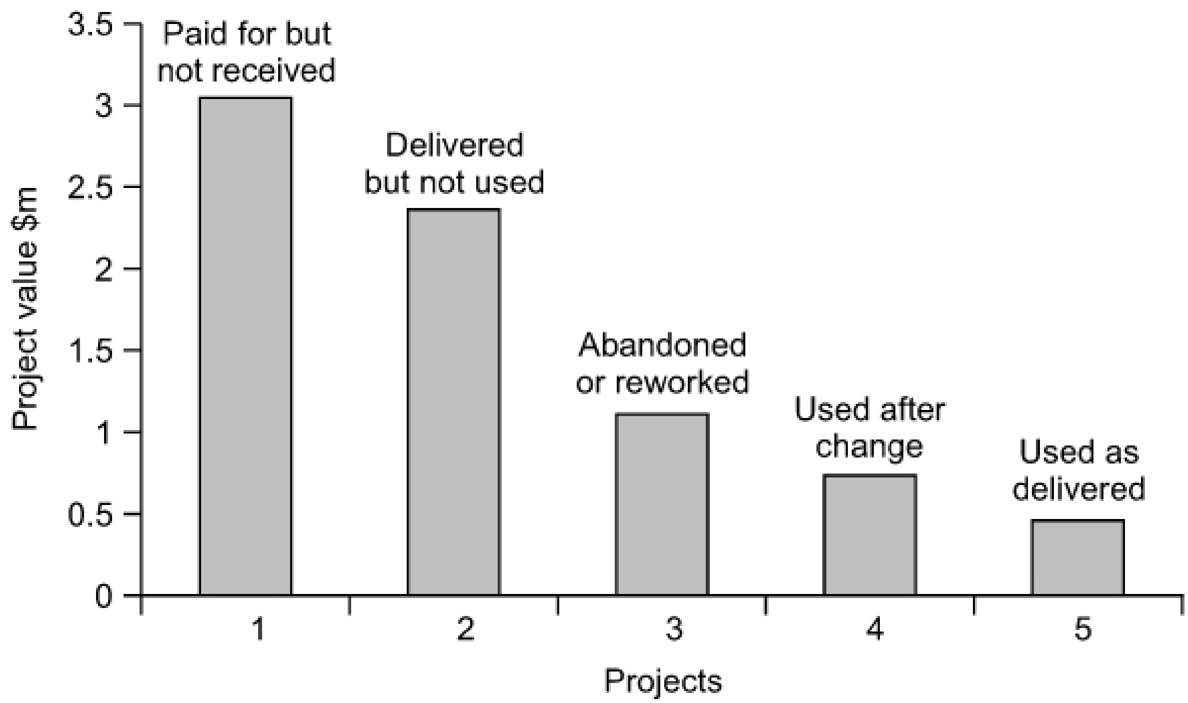


Fig. 1.1 *The state of US defence projects (according to the US government)*

data formats. It only illustrates that, in a changing world with a dynamic business environment, requests for change are unavoidable and therefore systems must be adaptable and tolerant to changes.

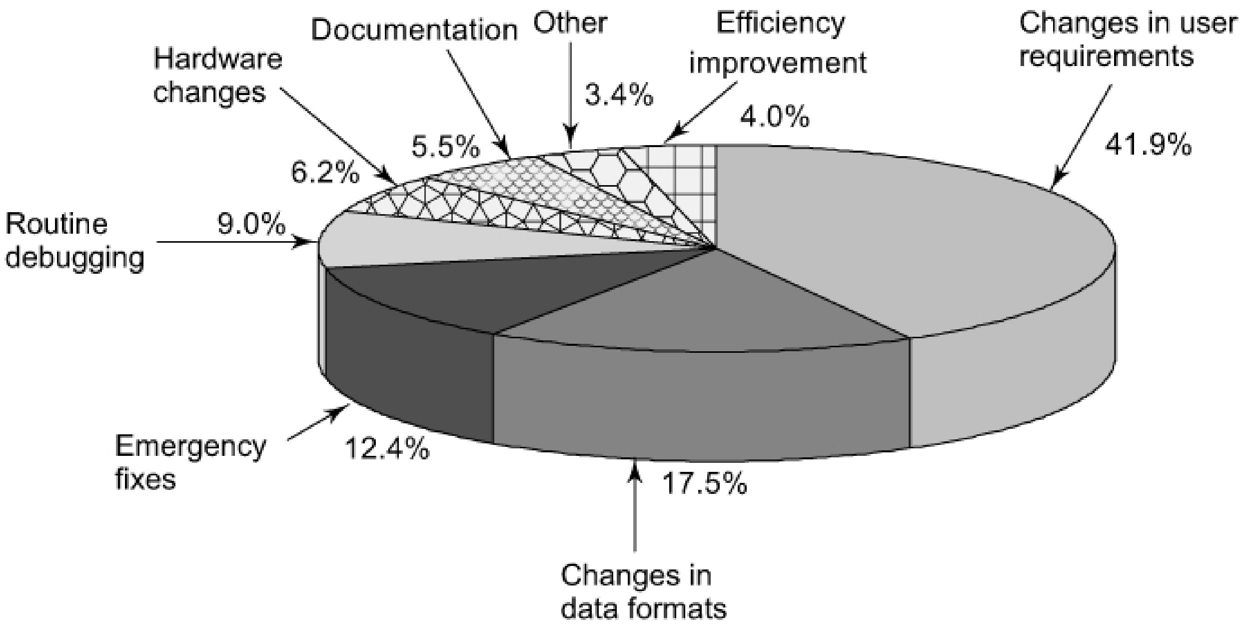


Fig. 1.2 *Breakdown of maintenance costs*

These studies and other reports on software implementation suggest that software products should be evaluated carefully for their quality before they are delivered and implemented. Some of the quality issues that must be considered for critical evaluation are:

1. Correctness
2. Maintainability
3. Reusability
4. Openness and interoperability
5. Portability
6. Security
7. Integrity
8. User friendliness

Selection and use of proper software tools would help resolving some of these issues.

1.2 Software Evolution

Ernest Tello, a well-known writer in the field of artificial intelligence, compared the evolution of software technology to the growth of a tree. Like a tree, the software evolution has had distinct phases or "layers" of growth. These layers were built up one by one over the last five decades as shown in Fig 1.3, with each layer representing an improvement over the previous one. However, the analogy fails if we consider the life of these layers. In software systems, each of the layers continues to be functional, whereas in the case of trees, only the uppermost layer is functional.

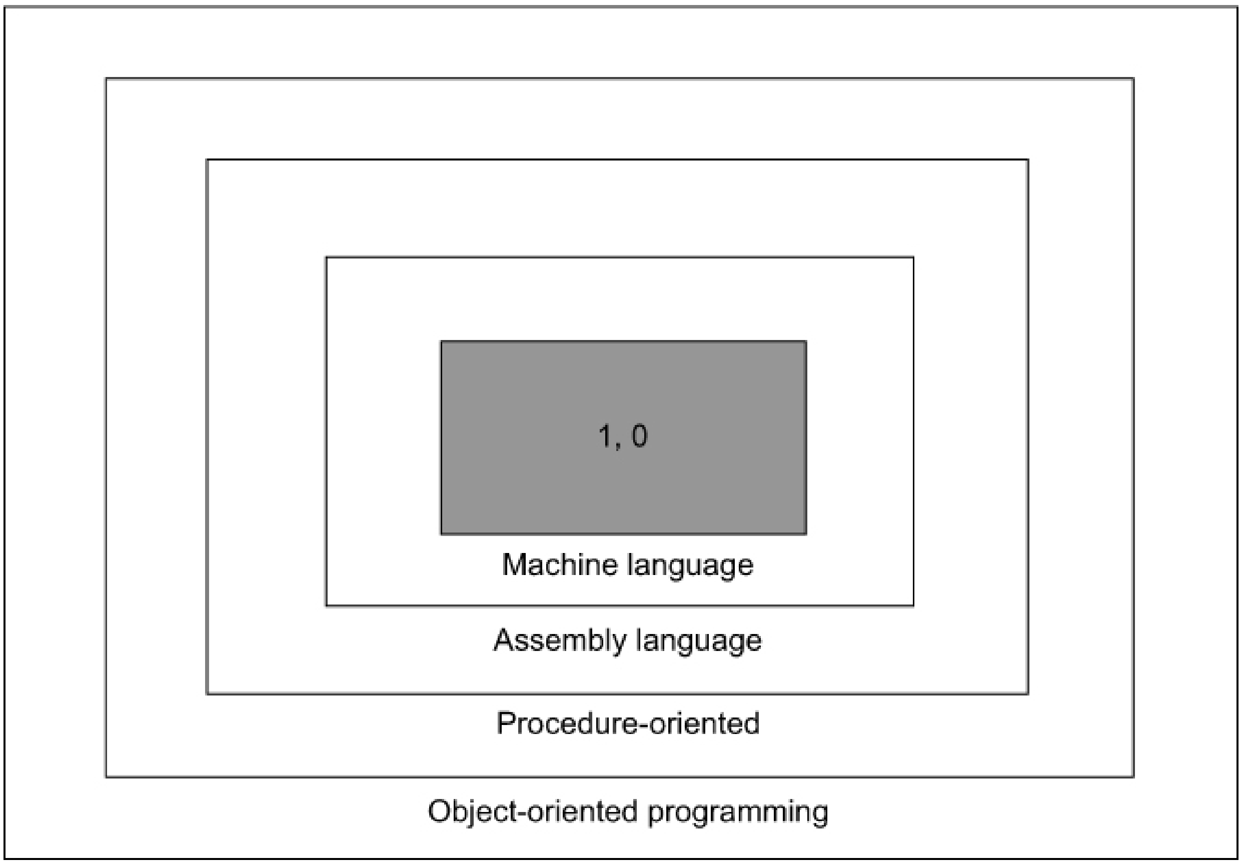


Fig. 1.3 *Layers of computer software*

Alan Kay, one of the promoters of the object-oriented paradigm and the principal designer of Smalltalk, has said: "/As *complexity increases, architecture dominates the basic material".* To build today's complex software it is just not enough to put together a sequence of programming statements and sets of procedures and modules; we need to incorporate sound construction techniques and program structures that are easy to comprehend, implement and modify.

Since the invention of the computer, many programming approaches have been tried. These include techniques such as *modular programming, top-down programming, bottom-up programming* and *structured programming.* The primary motivation in each has been the concern to handle the increasing complexity of programs that are reliable and maintainable. These techniques have become popular among programmers over the last two decades.

With the advent of languages such as C, structured programming became very popular and was the main technique of the 1980s. Structured programming was a powerful tool that enabled programmers to write moderately complex programs fairly easily. However, as the programs grew larger, even the structured approach failed to show the desired results in terms of bug-free, easy-to-maintain, and reusable programs.

Object-Oriented Programming (OOP) is an approach to program organization and development that attempts to eliminate some of the pitfalls of conventional programming methods by incorporating the best of structured programming features with several powerful new concepts. It is a new way of organizing and developing programs and has nothing to do with any particular language. However, not all languages are suitable to implement the OOP concepts easily.

1.3 A Look at Procedure-Oriented Programming

Conventional programming, using high level languages such as COBOL, FORTRAN and C, is commonly known as *procedure-oriented programming (POP).* In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions. A typical program structure for procedural programming is shown in Fig 1.4.

The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.

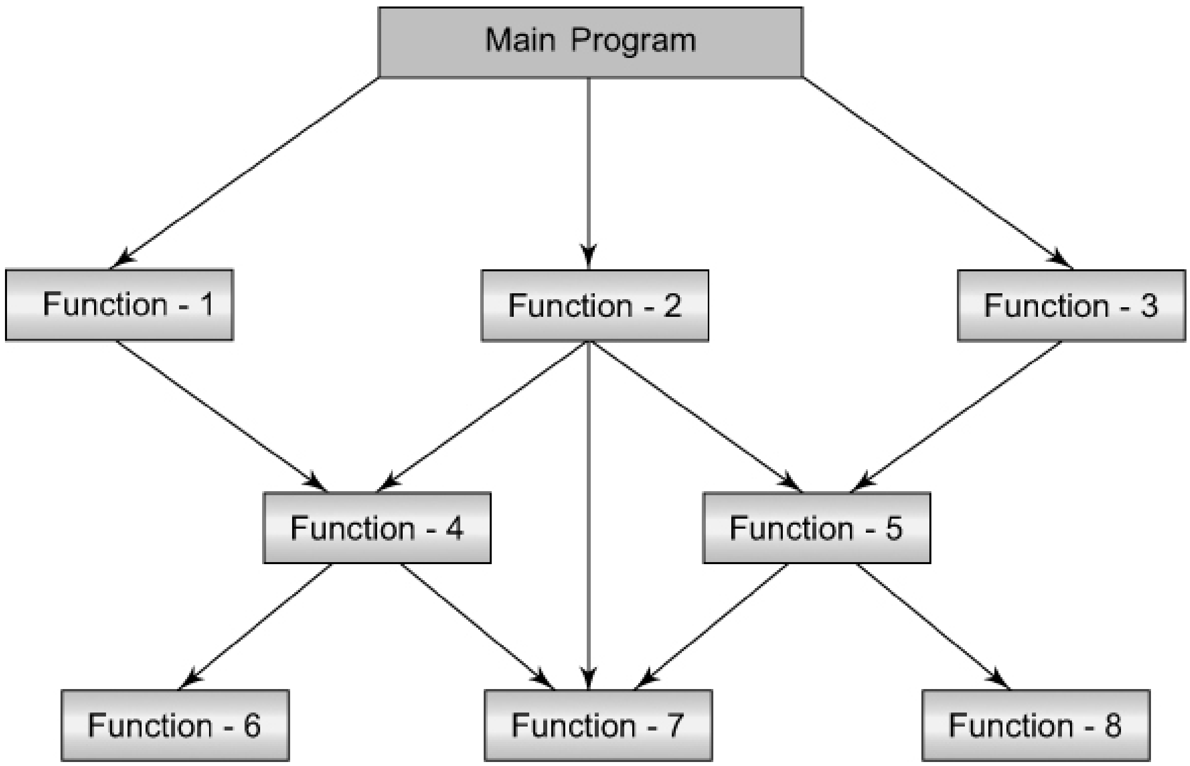


Fig. 1.4 *Typical structure of procedure-oriented programs*

Procedure-oriented programming basically consists of writing a list of instructions (or actions) for the computer to follow, and organizing these instructions into groups known as *functions.* We normally use a *flowchart* to organize these actions and represent the flow of control from one action to another. While we concentrate on the development of functions, very little attention is given to the data that are being used by various functions. What happens to the data? How are they affected by the functions that work on them?

In a multi-function program, many important data items are placed as *global* so that they may be accessed by all the functions. Each function may have its own *local data.* Figure 1.5 shows the relationship of data and functions in a procedure-oriented program.

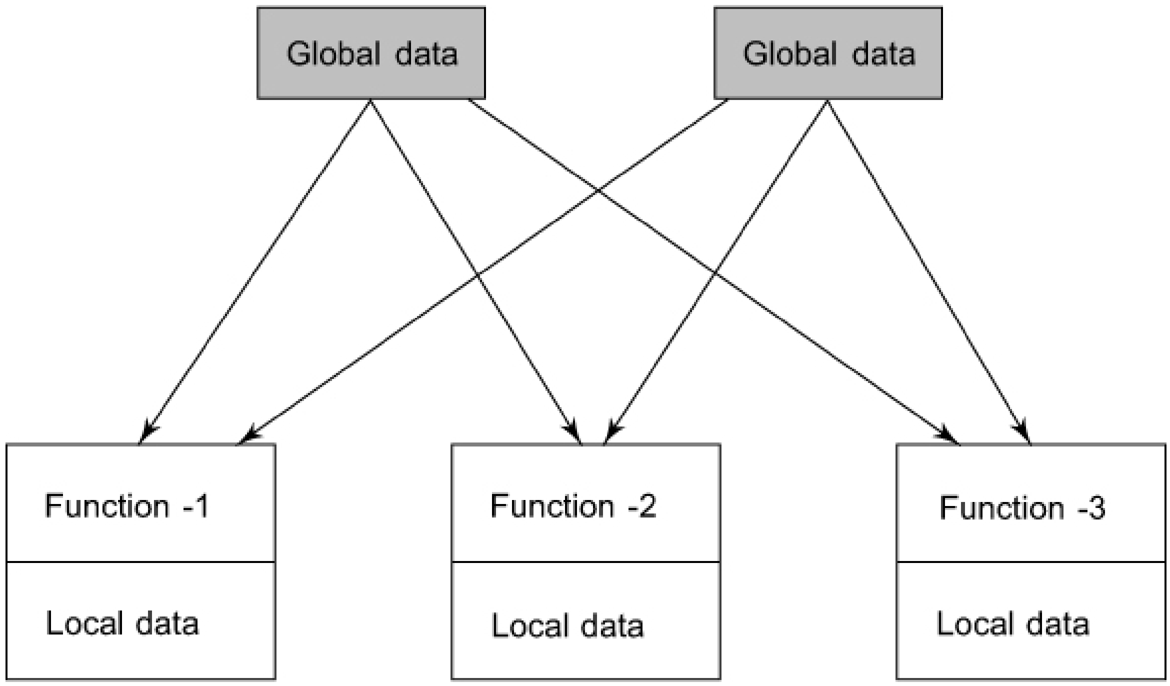


Fig. 1.5 *Relationship of data and functions in procedural programming*

Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

Another serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really correspond to the elements of the problem.

Some characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.

- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs *top-down* approach in program design.

1.4 Object-Oriented Programming Paradigm

The major motivating factor in the invention of object-oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called *objects* and then builds data and functions around these objects. The organization of data and functions in object-oriented programs is shown in Fig 1.6. The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the functions of other objects.

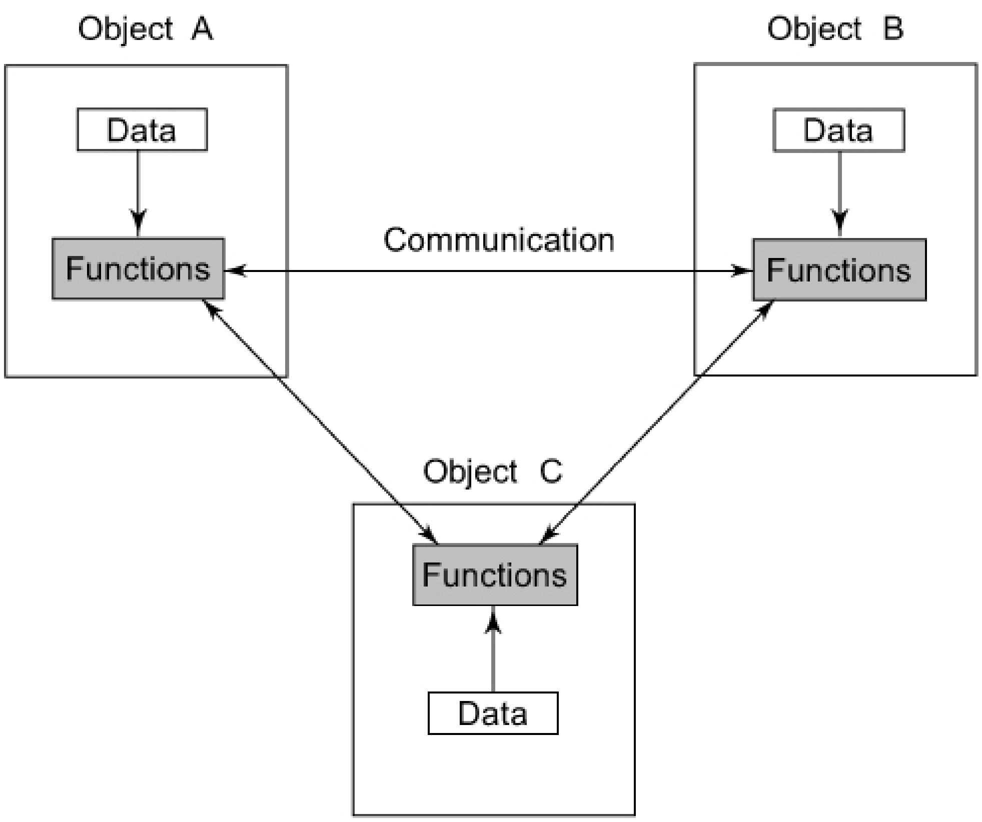


Fig. 1.6 *Organization of data and functions in OOP*

Some of the striking features of object-oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- *Follows bottom-up* approach in program design.

Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people. It is therefore important to have a working definition of object-oriented programming before we proceed further. We define "object-oriented programming as *an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand."* Thus, an object is considered to be a partitioned area of computer memory that stores data and set of operations that can access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

1.5 Basic Concepts of Object-Oriented Programming

It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

We shall discuss these concepts in some detail in this section.

Objects

Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in terms of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in C.

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object contains data, and code to manipulate the data. Objects can interact without having to know details of each other's data or code. It is sufficient to know the type of message accepted, and the type of response returned by the objects. Although different authors represent them differently, Fig 1.7 shows two notations that are popularly used in object-oriented analysis and design.

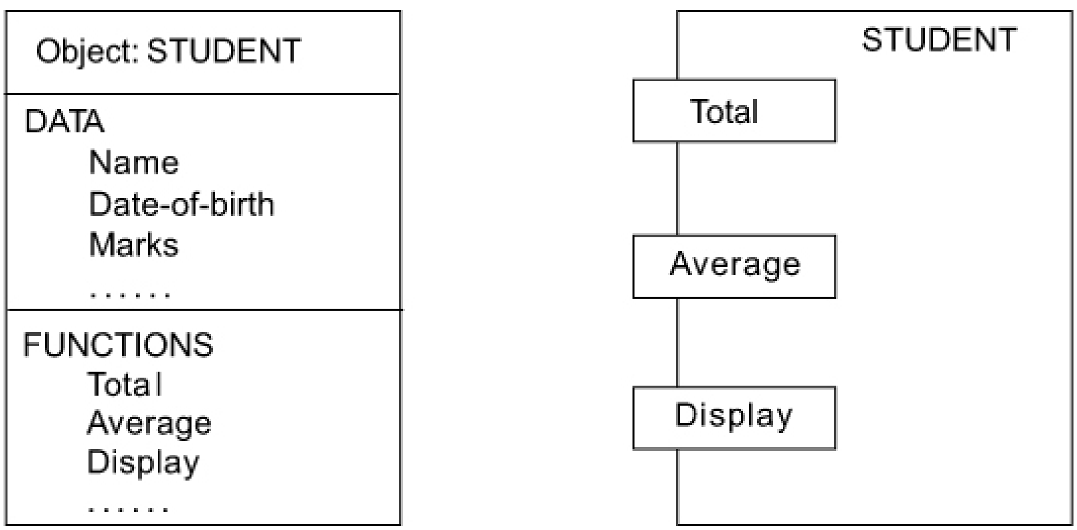


Fig. 1.7 *Two ways of representing an object*

Classes

We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a *class.* In fact, objects are variables of the type *class.* Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar type. For example, mango, apple and orange are members of the class fruit. Classes are user-defined data types and behave like the built-in types of a programming language. The syntax used to create an object is no different than the syntax used to create an integer object in C. If fruit has been defined as a class, then the statement

```
fruit mango;
```

will create an object **mango** belonging to the class **fruit.**

Data Abstraction and Encapsulation

The wrapping up of data and functions into a single unit (called class) is known as *encapsulation.* Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called *data hiding or information hiding.*

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract *attributes* such as size, weight and cost, and *functions* to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called *data members* because they hold information. The functions that operate on these data are sometimes called *methods or member functions.*

Since the classes use the concept of data abstraction, they are known as *Abstract Data Types* (ADT).

Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of *hierarchical classification.* For example, the bird 'robin' is a part of the class 'flying bird' which is again a part of the class 'bird'. The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in Fig 1.8.

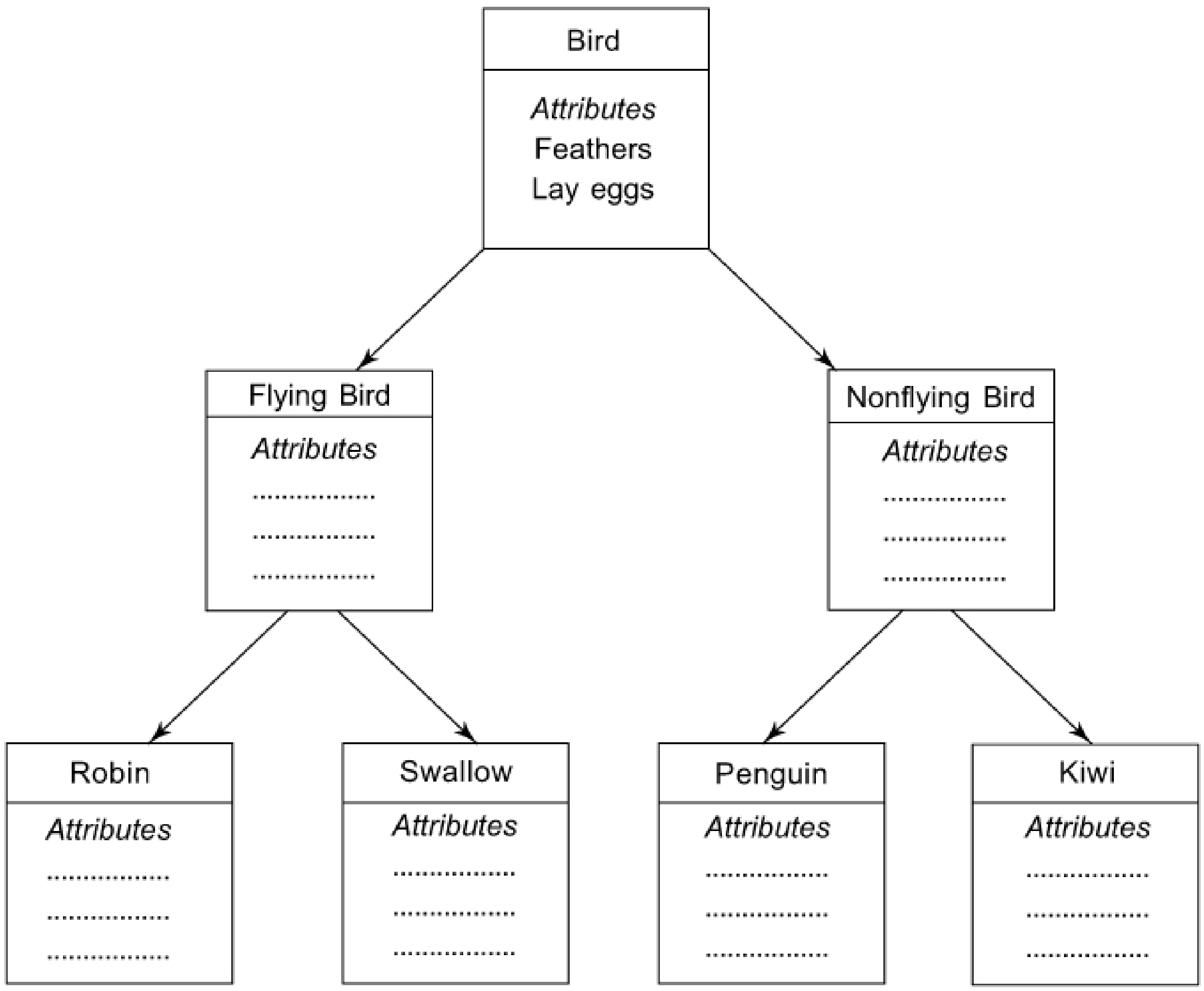


Fig. 1.8 *Property inheritance*

In OOP, the concept of inheritance provides the idea of *reusability.* This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side-effects into the rest of the classes.

Note that each sub-class defines only those features that are unique to it. Without the use of classification, each class would have to explicitly include all of its features.

Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviours in different instances is known as *operator overloading.*

Figure 1.9 illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context. Using a single function name to perform different types of tasks is known as *function overloading.*

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

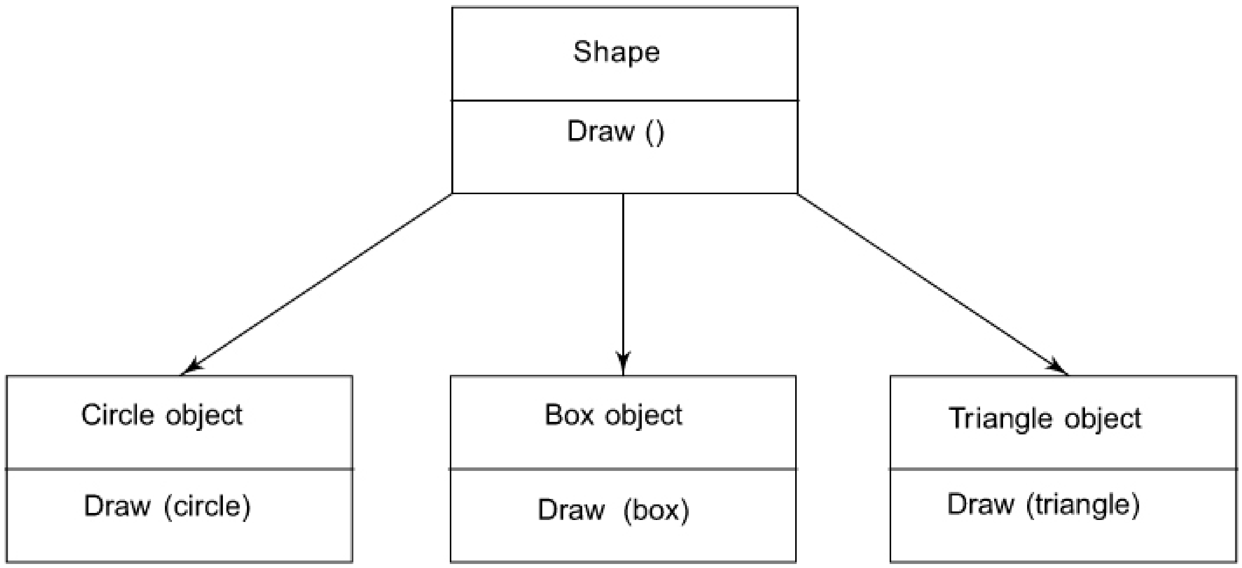


Fig. 1.9 *Polymorphism*

Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. *Dynamic binding* (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference.

Consider the procedure “draw” in Fig 1.9. By inheritance, every object will have this procedure. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be called.

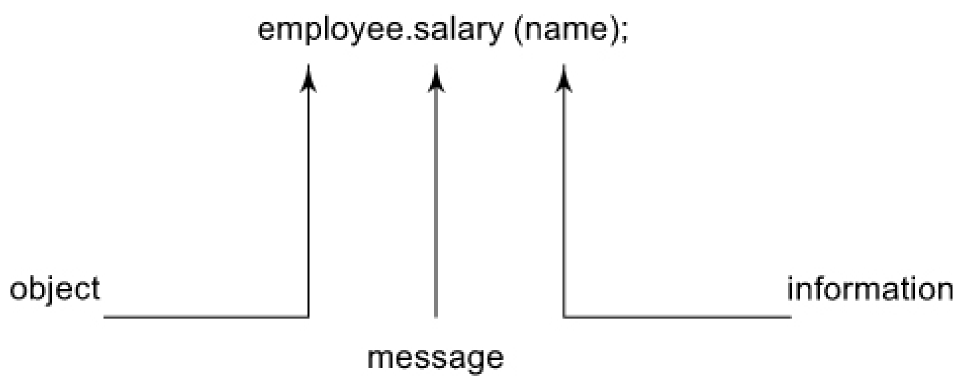
Message Passing

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:

1. Creating classes that define objects and their behaviour,
2. Creating objects from class definitions, and
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.

A message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired result. *Message passing* involves specifying the name of the object, the name of the function (message) and the information to be sent. Example:



Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is alive.

1.6 Benefits of OOP

OOP offers several benefits to both the program designer and the user. Object-orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map objects in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in imple-mentable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

While it is possible to incorporate all these features in an object-oriented system, their importance depends on the type of the project and the preference of the programmer. There are a number of issues that need to be tackled to reap some of the benefits stated above. For instance, object libraries must be available for reuse. The technology is still developing and current products may be superseded quickly. Strict controls and protocols need to be developed if reuse is not to be compromised.

Developing a software that is easy to use makes it hard to build. It is hoped that the object-oriented programming tools would help manage this problem.

1.7 Object-Oriented Languages

Object-oriented programming is not the right of any particular language. Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. However, programming becomes clumsy and may generate confusion when the programs grow large. A language that is specially designed to support the OOP concepts makes it easier to implement them.

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages, and
2. Object-oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major features that are required for object-based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to be object-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-oriented programming incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statement:

Object-based features + inheritance + dynamic binding

Languages that support these features include C++, Smalltalk, Object Pascal and Java. There are a large number of object-based and object-oriented programming languages. Table 1.1 lists some popular general purpose OOP languages and their characteristics.

Table 1.1 *Characteristics of some OOP languages*

Characteristics	*Simula* *	*Smalltalk* *	*Objective C*	*C++*	*Ada* **	*Object Pascal*	*Turbo Pascal*	*Eiffel* *	*Java* *
Binding	Both	Late	Both	Both	Early	Late	Early	Early	Both
(early or late)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Polymorphism	✓	✓	✓	✓	✓	✓	✓	✓	✓
Data hiding	✓	✓	✓	✓	✓	✓	✓	✓	✓
Concurrency	✓	Poor	Poor	Poor	Difficult	No	No	Promised	✓
Inheritance	✓	✓	✓	✓	No	✓	✓	✓	✓
Multiple Inheritance	No	✓	✓	✓	No	---	---	✓	No
Garbage Collection	✓	✓	✓	✓	No	✓	✓	✓	✓
Persistence	No	Promised	No	No	like 3GL	No	No	Some Support	✓
Genericity	No	No	No	✓	✓	No	No	✓	No
Object Libraries	✓	✓	✓	✓	Not much	✓	✓	✓	✓

* *Pure object-oriented languages*
** *Object-based languages*
Others are extended conventional languages

As seen from Table 1.1, all languages provide for polymorphism and data hiding. However, many of them do not provide facilities for concurrency, persistence and genericity. Eiffel, Ada and C++ provide generic facility which is an important construct for supporting reuse. However, persistence (a process of storing objects) is not fully supported by any of them. In Smalltalk, though the entire current execution state can be saved to disk, yet the individual objects cannot be saved to an external file.

Simula is one of the oldest object oriented programming languages, though it has spent most of its life in a research environment. Commercially, C++ and Java are the most prominent object oriented programming languages that have found extensive usage in application development.

Use of a particular language depends on characteristics and requirements of an application, organizational impact of the choice, and reuse of the existing programs. C++ has now become the most successful, practical, general purpose OOP language, and is widely used in industry today.

1.8 Applications of OOP

OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas. The most popular application of object-oriented programming, up to now, has been in the area of user interface design such as windows. Hundreds of windowing systems have been developed, using the OOP techniques.

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in these types of applications because it can simplify a complex problem. The promising areas for application of OOP include:

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems

- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

The richness of OOP environment has enabled the software industry to improve not only the quality of software systems but also its productivity. Object-oriented technology is certainly changing the way the software engineers think, analyze, design and implement systems.

Summary

- ❑ Software technology has evolved through a series of phases during the last five decades.
- ❑ The most popular phase till recently was procedure-oriented programming (POP).
- ❑ POP employs *top-down* programming approach where a problem is viewed as a sequence of tasks to be performed. A number of functions are written to implement these tasks.
- ❑ POP has two major drawbacks, viz. (1) data move freely around the program and are therefore vulnerable to changes caused by any function in the program, and (2) it does not model very well the real-world problems.
- ❑ Object-oriented programming (OOP) was invented to overcome the drawbacks of the POP. It employs the *bottom-up* programming approach. It treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it in a data structure called **class.** This feature is called **data** *encapsulation.*
- ❑ In OOP, a problem is considered as a collection of a number of entities called **objects.** Objects are instances of classes.

- Insulation of data from direct access by the program is called *data hiding.*
- *Data abstraction* refers to putting together essential features without including background details.
- *Inheritance* is the process by which objects of one class acquire properties of objects of another class.
- *Polymorphism* means one name, multiple forms. It allows us to have more than one function with the same name in a program. It also allows overloading of operators so that an operation can exhibit different behaviours in different instances.
- *Dynamic binding* means that the code associated with a given procedure is not known until the time of the call at run-time.
- *Message passing* involves specifying the name of the object, the name of the function (message) and the information to be sent.
- Object-oriented technology offers several benefits over the conventional programming methods—the most important one being the reusability.
- Applications of OOP technology has gained importance in almost all areas of computing including real-time business systems.
- There are a number of languages that support object-oriented programming paradigm. Popular among them are C++, Smalltalk and Java. C++ has become an industry standard language today.

Key Terms

Ada | assembly language | bottom-up programming | C++ | classes | concurrency | data abstraction | data encapsulation | data hiding | data members | dynamic binding | early binding | Eiffel | flowcharts | function overloading | functions | garbage collection | global data | hierarchical classification | inheritance | Java | late binding | local data | machine language | member functions | message passing | methods | modular programming | multiple inheritance | object libraries | Object Pascal | object-based programming | Objective C | object-oriented languages | object-oriented programming | objects | operator overloading | persistence | polymorphism | procedure-oriented programming | reusability | Simula | Smalltalk | structured programming | top-down programming | Turbo Pascal

Review Questions

1.1 What do you think are the major issues facing the software industry today?

1.2 Briefly discuss the software evolution during the period 1950 - 1990.

1.3 What is procedure-oriented programming? What are its main characteristics?

1.4 Discuss an approach to the development of procedure-oriented programs.

1.5 Describe how data are shared by functions in a procedure-oriented program.

1.6 What is object-oriented programming? How is it different from the procedure-oriented programming?

1.7 How are data and functions organized in an object-oriented program?

1.8 What are the unique advantages of an object-oriented programming paradigm?

1.9 Distinguish between the following terms:

(a) Objects and classes

(b) Data abstraction and data encapsulation

(c) Inheritance and polymorphism

(d) Dynamic binding and message passing

1.10 What kinds of things can become objects in OOP?

1.11 Describe inheritance as applied to OOP.

1.12 What do you mean by dynamic binding? How is it useful in OOP?

1.13 How does object-oriented approach differ from object-based approach?

1.14 List a few areas of application of OOP technology.

1.15 State whether the following statements are TRUE or FALSE.

(a) In procedure-oriented programming, all data are shared by all functions.

(b) The main emphasis of procedure-oriented programming is on algorithms rather than on data.

(c) One of the striking features of object-oriented programming is the division of programs into objects that represent real-world entities.

(d) Wrapping up of data of different types into a single unit is known as encapsulation.

(e) One problem with OOP is that once a class is created it can never be changed.

(f) Inheritance means the ability to reuse the data values of one object by another object.

(g) Polymorphism is extensively used in implementing inheritance.

(h) Object-oriented programs are executed much faster than conventional programs.

(i) Object-oriented systems can scale up better from small to large.

(j) Object-oriented approach cannot be used to create databases.